

SC200/SC300 Extra Software Programming Guide

Custom Property

1. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_AGC (204)

The property allows you to enable or disable TW6805's AGC loop circuit. When the AGC loop function is disabled, the AGC gain can be decided by your software manually. The range of AGC gain is from 0 to 511. If MSB bit.31 is set, the AGC loop function will be enabled.

SUPPORT VALUE: 0x00000000 ~ 0x000001FF - DISABLE AGC LOOP & SET AGC GAIN
SUPPORT VALUE: 0x80000000 - ENABLE AGC LOOP

EXAMPLE#01:

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 204, 0x80000000 );
```

EXAMPLE#02:

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 204, 0x00000100 );
```

2. KSPROPERTY_CUSTOM_XET_ANALOG_AUDIO_VOLUME (251)

The property allows you to adjust hardware's audio volume. The support range is from 0 to 255. 0 is mute.

SUPPORT VALUE: 0 ~ 255 - 0% ~ 100%

EXAMPLE#01:

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 251, 0 );
```

EXAMPLE#02:

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 251, 128 );
```

3. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_SWITCH_SPEED (205)

Software programmer can use this property to control the switching speed under switching mode^(*). Currently, there are 3 level speeds that can be controlled by you. Please reference this table as below: Here, the total fps means the total output frames per second for one chip under switching mode. For example, the total fps is 20fps. If you split one chip into four sub-channels, the every sub-channel's fps will be 5fps.

(*) Here, the switching mode means that one TW6805 chip is spitted to 2, 3 or 4 channels. We call these channels as sub-channel. SC300Q16 owns 4 chips and max 16 sub-channels.

RESOLUTION		SPEED	TOTAL FPS	COMMENT
D1	720×480	2	20FPS	
	704×480			
	640×480	1	20FPS	
	720×576			
	704×576	0	12FPS	
	640×576			
HALF D1	720×240	2	20FPS	
	704×240			
	640×240	1	30FPS	TO OBTAIN PERFECT OUTPUT RESULT, BUT TO CAUSE ONE LEFT-RIGHT SHIFTING SIDE EFFECT.
	720×288			
	704×288	0	15FPS	
	640×288			
CIF	360×240	2	20FPS	
	352×240			
	320×240	1	30FPS	TO OBTAIN PERFECT OUTPUT RESULT, BUT TO CAUSE ONE LEFT-RIGHT SHIFTING SIDE EFFECT.
	360×288			
	352×288	0	15FPS	
	320×288			

SUPPORT VALUE: 0, 1, 2

EXAMPLE#01:

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 205, 0 );
```

EXAMPLE#02:

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 205, 1 );
```

4. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_SWITCH_CHANNEL_TABLE (206)

In default setting, our switching algorithm uses one averaged channel table to control the channel switching sequence. The table size is 12 items length. Every item can be 0, 1, 2 or 3 to correspond to its sub-channels. For example, the split number is 4. The default switching channel table will be as { 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3 }.

Now, you can control the switching table dynamically by our SDK. For example, the table can be updated to { 0, 0, 1, 2, 0, 0, 1, 2, 0, 0, 1, 2 }. The total 20fps for every sub-channel will be changed as below:

CH#01: 10fps,
CH#02: 5fps,
CH#03: 5fps, and
CH#04: 0fps.

For another example, the table is { 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3 }. The result simulates one channel jumping effect.

Moreover, the table also can support single channel switching such as { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }. When the table is set, the switching mode will auto be returned to real-time mode. So, by this table, the CH#02's fps will be up to 30fps.

EXAMPLE#01: DISABLE CH#03.

```
BYTE TABLE[ 12 ] = { 0, 1, 3, 0, 1, 3, 0, 1, 3, 0, 1, 3 };  
AMESDK_SET_CUSTOM_PROPERTY_EX( hDev, 206, TABLE, 12 );
```

EXAMPLE#02: CHANNEL JUMPING.

```
BYTE TABLE[ 12 ] = { 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3 };  
AMESDK_SET_CUSTOM_PROPERTY_EX( hDev, 206, TABLE, 12 );
```

EXAMPLE#03: GET CURRENT SWITCH CHANNEL TABLE.

```
BYTE TABLE[ 12 ];  
AMESDK_GET_CUSTOM_PROPERTY_EX( hDev, 206, TABLE, 12 );
```

EXAMPLE#04: SINGLE CHANNEL OUTPUT.

```
BYTE TABLE[ 12 ] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };  
AMESDK_SET_CUSTOM_PROPERTY_EX( hDev, 206, TABLE, 12 );
```

5. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_SWITCH_RESOLUTION_TABLE (207)

The custom property allows the developer to change default resolution at any time. For example, at four channel's switching mode, developer need obtain this configuration as below:

CH#01: 704x480

CH#02: 704x240

CH#03: 352x240

CH#04: 352x240

The property programming is similar to **ANALOG_VIDEO_SWITCH_CHANNEL_TABLE**. It also uses 12 bytes to setup the kernel driver's resolution table. Every item is corresponded to its channel item in the switching channel table. The parameter range is from 0 to 2.

0x00: D1

0x01: HALF.D1

0x02: CIF.

EXAMPLE#01: SETUP RESOLUTION TABLE AS BELOW:

CH#01: 704x480

CH#02: 704x240

CH#03: 352x240

CH#04: 352x240

```
BYTE CHANNEL_TABLE[ 12 ] = { 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3 };
```

```
BYTE RESOLUTION_TABLE[ 12 ] = { 0, 1, 2, 2, 0, 1, 2, 2, 0, 1, 2, 2 };
```

```
AMESDK_SET_CUSTOM_PROPERTY_EX( hDev, 206, CHANNEL_TABLE, 12 );
```

```
AMESDK_SET_CUSTOM_PROPERTY_EX( hDev, 207, RESOLUTION_TABLE, 12 );
```

6. KSPROPERTY_CUSTOM_XET_ANALOG_VIDEO_POST_FRAME_RATE (208)

Generally, the software developer can use AMESDK_SET_FORMAT to control video frame rate output. For some special applications, developer could adjust the frame rate dynamically during recording. The post frame rate is dynamically used to adjust current stream output, which is set by AMESDK_SET_FORMAT function at initialize stage. The range of the post frame rate property is from 0 to 255. It is identical to the skip number of frame (or field). The value 1 means the recording frame rate is 15.000fps in NTSC.

EXAMPLE#01: SET FRAMERATE TO 30.000FPS:

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 208, 0 ); // 30FPS / (0 + 1) = 30FPS
```

EXAMPLE#02: SET FRAMERATE TO 15.000FPS:

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 208, 1 ); // 30FPS / (1 + 1) = 15FPS
```

EXAMPLE#03: SET FRAMERATE TO 10.000FPS:

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 208, 2 ); // 30FPS / (2 + 1) = 10FPS
```

7. KSPROPERTY_CUSTOM_XET_GPIO_DIRECTION (940)

7. KSPROPERTY_CUSTOM_XET_GPIO_DATA (941)

The property allows you to access TW6805's GPIO interface. The property KSPROPERTY_CUSTOM_XET_GPIO_DIRECTION allows you to control its direction. Here, writing 1 to bit enables this pin as output pin. Usually, the GPIO is controlled by the first chipset in one board.

SUPPORT VALUE: 0 ~ 1 - INPUT ~ OUTPUT

The property KSPROPERTY_CUSTOM_XET_GPIO_DATA allows you to access GPIO's data.

SUPPORT VALUE: 0 ~ 1 - LOW ~ HIGH

EXAMPLE#01: TO DEFINE GPIO AS 8 OUTPUT PINS [0:7] AND 8 INPUT PINS [8:15].

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 940, 0x00FF );
```

EXAMPLE#02: TO DEFINE GPIO AS 16 OUTPUT PINS [0:15] THEN PULL HIGH FOR ALL.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 940, 0xFFFF );
```

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 941, 0xFFFF );
```

EXAMPLE#03: TO DEFINE GPIO AS 16 INPUT PINS [0:15] THEN READ DATA FROM IT.

```
AMESDK_SET_CUSTOM_PROPERTY( hDev, 940, 0x0000 );
```

```
AMESDK_GET_CUSTOM_PROPERTY( hDev, 941, &GPIO );
```

8. Application Note for AMESDK_GET_LOCK()

Customer to use AMESDK_GET_LOCK, please notes it. If your card is N series, the return value is described by 1 bit only. High is signal lock, and low is unlock. If your card is D series, the return value will use 2 bits to describe both sub-channels' status. If card is Q series, we will use 4 bits to describe all sub-channels.

EXAMPLE#01: GET SC300N4 SIGNAL STATUS.

```
AMESDK_GET_LOCK( hDev[ 0 ], &status[ 0 ] ); // GET CH01 STATUS
AMESDK_GET_LOCK( hDev[ 1 ], &status[ 1 ] ); // GET CH02 STATUS
AMESDK_GET_LOCK( hDev[ 2 ], &status[ 2 ] ); // GET CH03 STATUS
AMESDK_GET_LOCK( hDev[ 3 ], &status[ 3 ] ); // GET CH04 STATUS
```

EXAMPLE#02: GET SC300D8 SIGNAL STATUS.

```
AMESDK_GET_LOCK( hDev[ 0 ], &status[ 0 ] ); // GET CH01 ~ CH02 STATUS
AMESDK_GET_LOCK( hDev[ 1 ], &status[ 1 ] ); // GET CH03 ~ CH04 STATUS
AMESDK_GET_LOCK( hDev[ 2 ], &status[ 2 ] ); // GET CH05 ~ CH06 STATUS
AMESDK_GET_LOCK( hDev[ 3 ], &status[ 3 ] ); // GET CH07 ~ CH08 STATUS
```

EXAMPLE#03: GET SC300Q16 SIGNAL STATUS.

```
AMESDK_GET_LOCK( hDev[ 0 ], &status[ 0 ] ); // GET CH01 ~ CH04 STATUS
AMESDK_GET_LOCK( hDev[ 1 ], &status[ 1 ] ); // GET CH05 ~ CH08 STATUS
AMESDK_GET_LOCK( hDev[ 2 ], &status[ 2 ] ); // GET CH09 ~ CH12 STATUS
AMESDK_GET_LOCK( hDev[ 3 ], &status[ 3 ] ); // GET CH13 ~ CH16 STATUS
```


9. Access Custom Property for DirectShow Developer

Customer uses DirectShow to develop software can bypass our SDK to access TW6805 directly. All custom properties are implemented by IKsPropertySet interface. The interface can be queried from our capture source filter.

EXAMPLE#01: SET SWITCHING SPEED.

```
static const GUID GUID_KPS_TW6805 = { 0xD1E5209F, 0x68FD, 0x4529, 0xBE, 0xE0, 0x5E, 0x7A, 0x1F, 0x47, 0x92, 0x12 };
ULONG speed = 0;
m_pKsPropertySet->Set( GUID_KPS_TW6805, 205, NULL, 0, &speed, sizeof(ULONG) );
```

10. Application Note for DirectShow Developer

The developer who uses DirectShow to access our capture source filter need check the frame size in the callback function of your SampleGrabber class. If the frame size is 0 bytes, it means the frame is one bad frame. You should drop it. More detail, please check with our engineer team directly.